MatSWMM

SWMM 5.1 Co-simulation Toolbox

Universidad de los Andes - Department of Electrical and Electronic Engineering

Contents

1 Introduction

SWMM is an open source platform that is used for planning, analysis and design of related to stormwater runoff, combined and sanitary sewers, and other drainage systems in urban areas. The program allows its users, through an interface, to model a drainage network, using objects such as conduits, storage units, subcatchments, among others. The user sets attributes for each element in the network and then, executes the calculation module of SWMM.

Due to the changes that the design of sewer systems is suffering today, because of the inclusion of actuators, control rules, and optimal design, SWMM has become a limited tool; despite of its rule-based control module. However, it has a big potential being an open source software. Thanks to this, it is possible to enhance its functionality in order to include new methods that allow its users to simulate optimization and control processes.

Because of this, MatSWMM has been developed in order to bring new functionalities to the SWMM framework. This toolbox works as a co-simulation engine that retrieves information from SWMM during the simulation and allows to modify initial conditions, and attributes like an actuator setting or the maximum volume of an storage unit.

Furthermore, in order to make this tool scalable, it has been developed for three programming languages: Python, Matlab, and LabVIEW. The syntax for each tool is very similar, so it is easy to migrate from one to another depending on the type of project is going to be developed. It is worth to say that this toolbox requires little effort to be learnt if the user has worked with SWMM before. Nevertheless, some knowledge about the framework, and the programming language is required.

That said, this document will cover five main topics:

1. *How does SWMM work?* The main goal of this section is to clarify doubts related to files that are required to run a SWMM simulation. Also, explain the calculation algorithm of SWMM, which is necessary to understand how the toolbox functionalties work, and how to use them for a co-simulation project with MatSWMM.

2. *How does the MatSWMM work?* A general description of the toolbox is given.

3. *Co-simulation functions and constants* A Deep explanation of each of the functions and constants included in the toolbox.

4. *Examples* A case study is developed for the three programming languages in order to compare the ways to declare the main functionalities of MatSWMM.

2 How does SWMM work?

For the purpose of this manual, the complete explanation of the computational methods and the base model used by SWMM is not going to be explained; it can be checked in the SWMM 5 User's Manual which is available on the https://goo.gl/YBD3SsSWMM Website. However, it is important to note that SWMM is a phisically based, discrete-time simulation model which is based on principles of conservation of mass, energy, and momentum. As a discrete-time tool its algorithm finds n solutions, for the flow routing problem in n time intervals, for a time window and initial conditions defined by the user. With that in mind, it is possible to understand how SWMM makes its calculations checking the whole process; there are seven subprocesses which make up the simulation algorithm as it is shown in Figure 1.

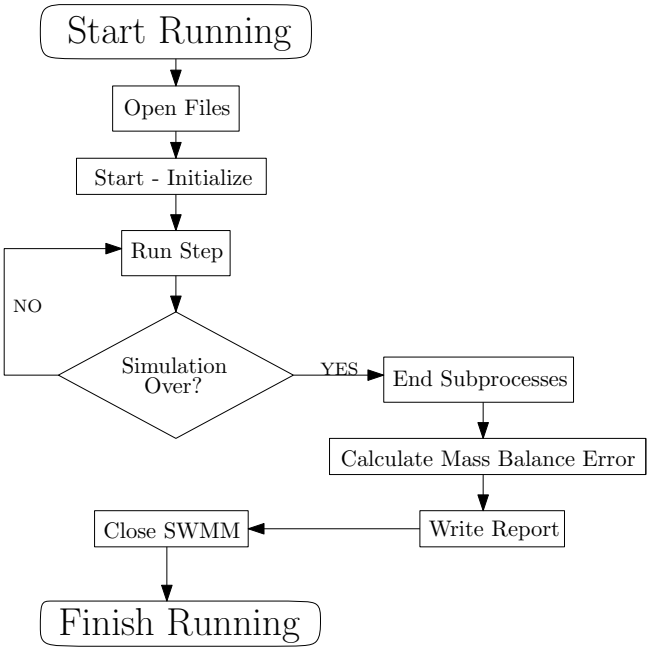


Figure 1. Algorithm to run a simulation in SWMM.

1. Open SWMM files: SWMM opens three files along the simulation, the input file `.inp` that contains all the information about the model, the report file `.rpt` which contains a status report of the results, and the output file `.out` that is a binary file with the numerical results of a successful simulation.

2. Start simulation: the internal parameters of the program are initialized.

3. Run step: advances the simulation by one routing time step.
4. End subprocesses: all the computing systems and subprocesses of SWMM are ended.
5. Calculate mass balance error: at the end of a simulation SWMM calculates three errors based on the mass conservation principle, the runoff error, the flow routing error, and the quality routing error.
6. Write report: SWMM writes the information for the output and report files.
7. Close SWMM: the files used by SWMM are closed. The simulation is over.

3 How does MatSWMM work?

MatSWMM has been created as an additional module of the SWMM computational engine in order to preserve the code integrity. It has been compiled as a DLL, so it can be compatible with C-based programming languages. Additionally, some of the DLL functions that were created by the US-EPA were reused. Taking this into consideration, in order to enhance the functionality of MatSWMM, following the style that has been used so far, you can download the original code of SWMM from the SWMM Website and the toolbox C files from the Toolbox Repository.

4 Co-simulation functions and constants

MatSWMM is composed of several functions that can reproduce a SWMM simulation with control actions and optimal model attributes. It also has useful constants that simplify the code development process. In Tables 2, 1, 4, and 3 both general functions and constants are described deeply. The functionalities of MatSWMM have been classified as:

- Basic functionalities: these are inherited from the SWMM framework, and allow to reproduce the SWMM algorithm.
- Getters and setters: these are enhanced functionalities, that can be used to extract information from the SWMM model, or modify the setting of actuators, during the simulation.
- Data management functionalities: these allow to manage large data sets efficiently, with graph-based traversal algorithms.

Table 1. Getters and setters included in MatSWMM.

Function	Description
`step_get`	It retrieves the values of an specific property of multiple objects while running the simulation.
`get_from_input`	It returns the value of the attribute in the "`.inp`" file.
`get_all`	It returns all the objects IDs of a certain type (e.g., NODES, LINK, SUBCATCH, STORAGE, OUTFALL, JUNCTION) and the value of one of their properties.
`modify_input`	It modifies a specific attribute from the "`.inp`" file.
`modify_settings`	It modifies the setting of several orifices during the simulation.

Table 2. MatSWMM basic functionalities.

Function	Description
`initialize`	It opens the files required to run a SWMM simulation, and starts it.
`run_step`	It advances the simulation by one routing time step.
`is_over`	It determines whether or not the simulation is over.
`finish`	It saves all the results of the simulation and closes the program to run a new simulation.
`run_simulation`	It runs a SWMM simulation.

Table 3. Data management functionalities.

Function	Description
`get_incidence_matrix`	It returns the graph representation of the network in matrix form.
`save_results`	It saves all the results of the simulation in "`.csv`" files, organized in four folders in the workspace directory. The folders are related to the type of objects (Link, Node, Subcatch). A folder called *Time* with information of the step size is also saved.
`create_graph`	It creates a graph data structure in Python that represents the network as a connection of nodes (manholes) and conduits (canals and pipes).

Table 4. Constants with their descriptions.

Types of objects	
SUBCATCH	Subcatchment object
NODE	Node Object
LINK	Link object
Unit System	
US	English unit system
SI	International unit system
DIMENSIONLESS	Used if an attribute is dimentionless (e.g. Froude number)
Type of Attributes	
DEPTH	Max. depth of links and nodes.
VOLUME	Volumetric capacity of links and nodes.
FLOW	Link flow.
SETTING	Orifice setting - percentage of openning (decimal number).
FROUDE	The Froude number in a link object.
INFLOW	The inflow of a node object.
FLOODING	Flood volume of a node.
PRECIPITATION	Intensity of precipitation in a subcatchment.
RUNOFF	Runoff in a subcatchment.
Report Options	
NO_REPORT	The simulation results are not written in the report file.
WRITE_REPORT	The simulation results are written in the report file.
Input file	
INVERT	Invert elevation of a node.
DEPTH_SIZE	Depth of links, junctions and storage units.
STORAGE_A	A - Geometric parameter of an storage unit.
STORAGE_B	B - Geometric parameter of an storage unit.
STORAGE_C	C - Geometric parameter of an storage unit.
LENGTH	The length of a link.
ROUGHNESS	Roughness of a link.
IN_OFFSET	Inlet offset of a conduit.
OUT_OFFSET	Outlet offset of a conduit.
JUNCTION	Junction object (It belongs to the node type)
STORAGE	Storage object (It belongs to the node type)

5 Examples

The first case study

The model used for this example is available in the Toolbox Repository. Basically, it is composed by four subcatchments that are affected by a precipitation event, the precipitation becomes runoff, and it is transported through a system of channels that is divided in three (see Figure 2. At the beginning of the bifurcation there are three gates, modelled as orifices, controlling the passage of rainwater in order to distribute it efficiently in three tanks of different volumes.

In this example it is going to be shown how to use the functions of the Toolbox to reproduce the Running Algorithm of SWMM, adding the functionality of retrieving and modifying parameters during the simulation.

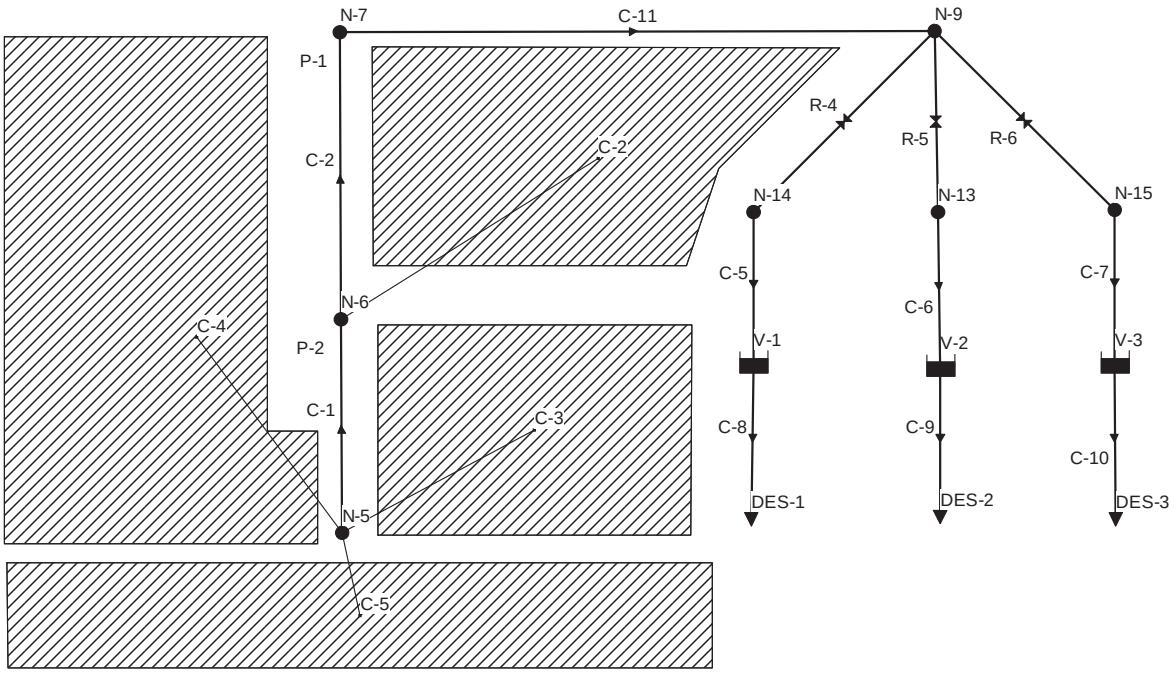


Figure 2. First case study used as an example to show the functionalities of MatSWMM.

The second case study

This case study is a large scale drainage network with full load (i.e., all the receptors and conduits are used). The whole system is shown in Figure 3. It is used to show the capabilities of MatSWMM with large-scale systems, since it extracts information from the model efficiently.

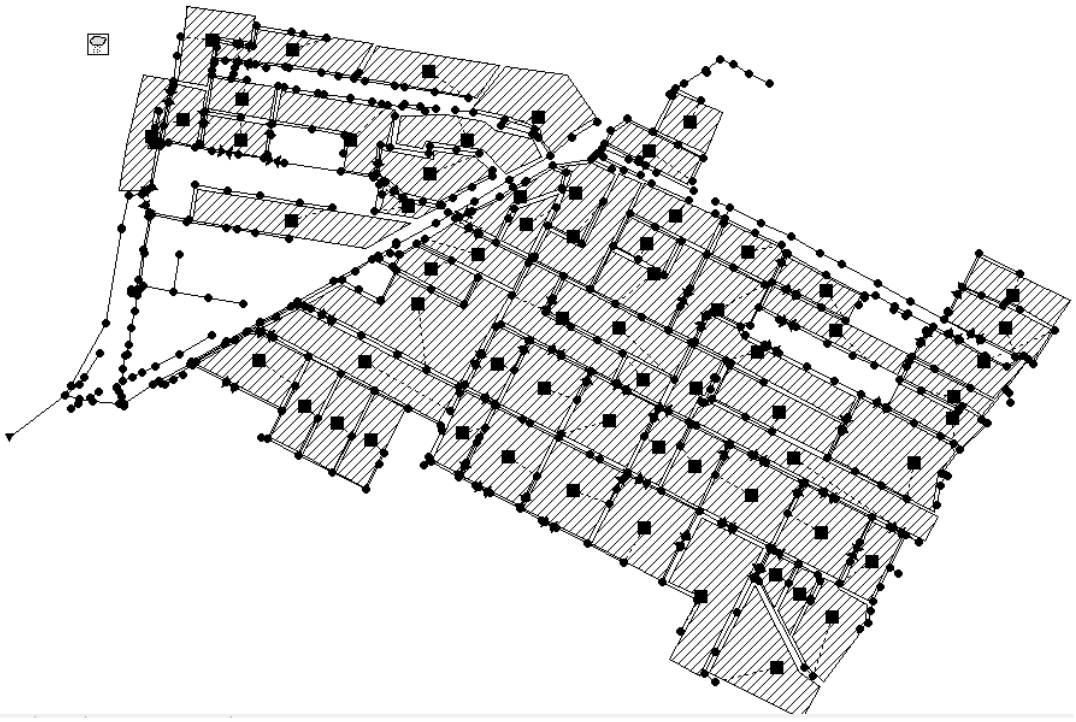


Figure 3. Second case study used as an example to show the functionalities of MatSWMM

The Python Toolbox - (Check the source code here)

```
# Imported modules
import swmm  # The SWMM module
import matplotlib.pyplot as plt  # Module for plotting

# ********************************************************************
#  Declaration of simulation files and variables
```

```
# ****************************************************************************

inp    = "swmm_files/3tanks.inp"  # Input file
flow   = []
vol    = []
time   = []

# ****************************************************************************
#  Initializing SWMM
# ****************************************************************************

swmm.initialize(inp)  # Step 1

# ****************************************************************************
#  Step Running
# ****************************************************************************

# Main loop: finished when the simulation time is over.
while( not swmm.is_over() ):

  # ----------------- Run step and retrieve simulation time -----------

  time.append( swmm.get_time() )
  swmm.run_step()  # Step 2

  # --------- Retrieve & modify information during simulation ---------
  # Retrieve information about flow in C-5
  f = swmm.get('C-5', swmm.FLOW, swmm.SI)
  # Stores the information in the flow vector
  flow.append(f)
  # Retrieve information about volume in V-1
  v = swmm.get('V-1', swmm.VOLUME, swmm.SI)
  # Stores the information in the volume vector
  vol.append(v)

  # --------------------------- Control Actions ------------------------

  # If the flow in C-5 is greater or equal than 2 m3/s the setting
  # upstream of the link is completely closed, else it is completely
  # opened.

  if f >= 2:
    swmm.modify_setting('R-4', 0)
  else:
    swmm.modify_setting('R-4', 1)
```

```
# ****************************************************************************
#  End of simulation
# ****************************************************************************

errors = swmm.finish() # Step 3

# ****************************************************************************
#  Interacting with the retrieved data
# ****************************************************************************

print "\n Runoff error: %.2f %%\n \
  Flow routing error: %.2f %%\n \
  Quality routing error: %.2f %%\n" % (errors[0], errors[1], errors[2])

plt.plot(time, flow)
plt.title('C-5 flow [m3/s]')
plt.xlabel('Time [hours]')
plt.show()
```

After checking the code you can notice that it has been divided in sections, so it is possible to relate each section to a part of the Running Algorithm. Additionally, every step related to the Running Algorithm has been marked with its corresponding number; it is important to respect this order. It is also important to be aware of the functions and constants declaration. In this case, all the constants and functions of the SWMM module have been imported. However, if the module had been imported as a single component, all the functions should have been declared instantiating the module.

```
swmm.start(swmm.NO_REPORT) # e.g.
```

Now, checking the code in more detail there are additional functions related to the Python Toolbox:

- The *is_over*() function: it has been created in order to determine when the simulation is over. It returns *True* if there are no more steps left to run.
- The *get_time*() function: it returns the current simulation time in hours. It is important to notice that is has been declared before the *run_step*() function. This is not a coincidence, it is important to do this if you want to get a coherent time vector at the end of the simulation. If you declare it after *run_step*(), in the last step the time value is going to be zero, because zero is the value generated by *run_step*() to warn that the simulation is over.

Besides this, the syntax to retrieve and modify information is pretty simple and the control actions can be as complicated as Python coding is. The last important thing is that the *get_mass_bal_error*() function returns a tuple with the values of each error. The first value in the tuple is the runoff error, the second is the flow routing error and the last is the quality routing error.

The results obtained running the example code are shown in Figure 4. As you can see, the rule-based control action modifies the valve setting maintaining the flow in C-5 is relatively close to 2 m^3/s.

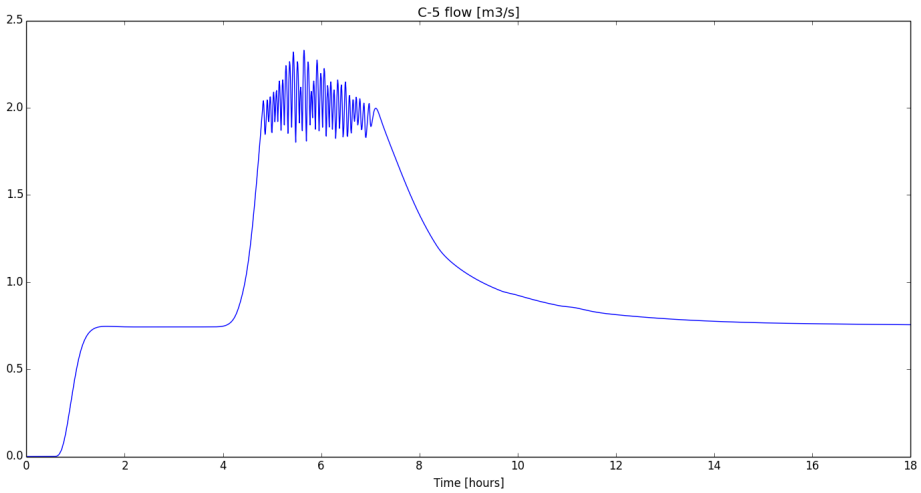


Figure 4. Python example results.

The Matlab Toolbox - (Check the source code here)

```
%% INITIALIZE SWMM
    inp = 'swmm_files/redchicosur_gates.INP';
    swmm = SWMM;
%% RETRIEVING VARIABLE IDs FROM THE .INP
    links = swmm.get_all(inp, swmm.LINK, swmm.NONE);
    nodes = swmm.get_all(inp, swmm.NODE, swmm.NONE);
%% RUNNING A SWMM SIMULATION
    [e, d] = swmm.run_simulation(inp);
%% RETRIEVING INFORMATION
```

```
fprintf('Total flooding = %.3f m3\n', swmm.total_flooding);
[time, fn5] = swmm.read_results(links, swmm.LINK, swmm.VOLUME);
plot(time, fn5);

disp('MatSWMM is awesome! isnt it?');
```

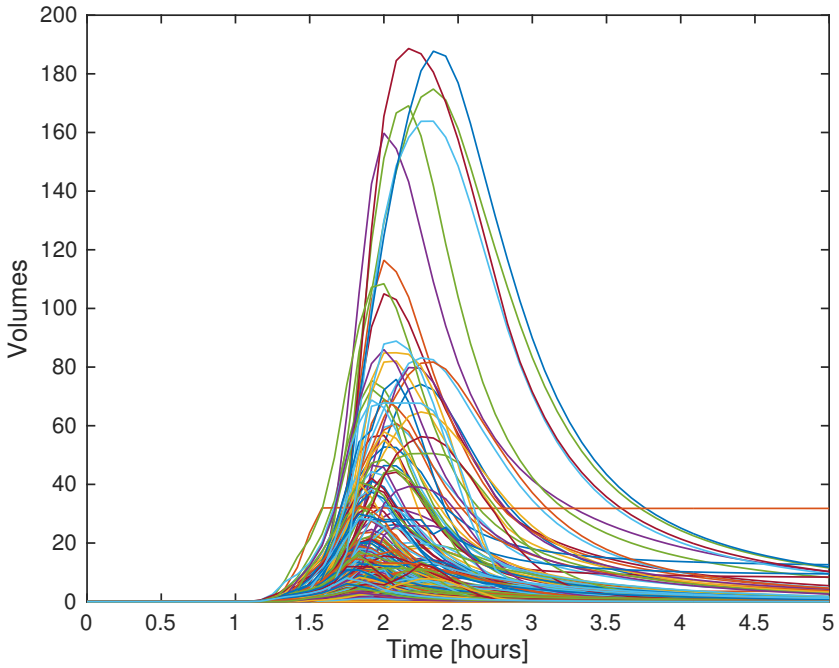


Figure 5. Python example results.

The Matlab example code has been divided in sections so each section is related with a part of the SWMM Running Algorithm. Additionally, each step of the Running Algorithm has been marked and enumerated; it is necessary to respect this order. However there are special functions and variables for the Matlab module that have not been specified yet, these are:

- The *read_results* function is used to read efficiently simulation results that have been stored in `.csv` files, at the Matlab's current workspace.

- The *total_flooding* function is used to compute the total flooding of the whole network.

Finally, it is worth saying that the *run_simulation* function returns the errors, related to the mass balance testing, as a vector with three elements. The first one is the runoff error, the second is the flow routing error and the third is the quality error.

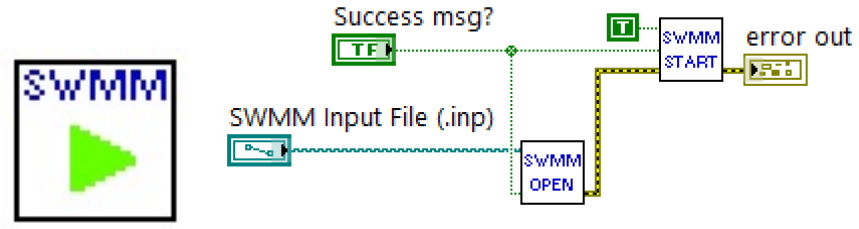
The results obtained after the simulation are shown in Figure 5. These are the volumes stored in each of the conduits and reservoirs, which are more than five hundred, during the simulation time.

The LabVIEW toolbox

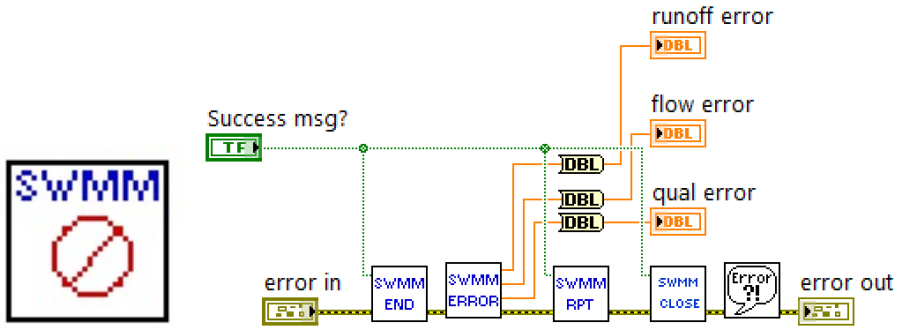
Differently to the Python and Matlab toolboxes, the functionalities of the LabVIEW toolbox have been developed using blocks with inputs and outputs. For the LabVIEW package it is possible to reproduce a SWMM simulation, extract parameters, and modify the setting of actuators.

Some of the functionalities that have been described thorough the document are described below for the LabVIEW environment:

- `initialize`:



- `stop`:



- `get`:

- `step`:

- `modify_setting`:

An example of the structure of a LabVIEW program for running a simulation and extracting information from the SWMM model is shown below:

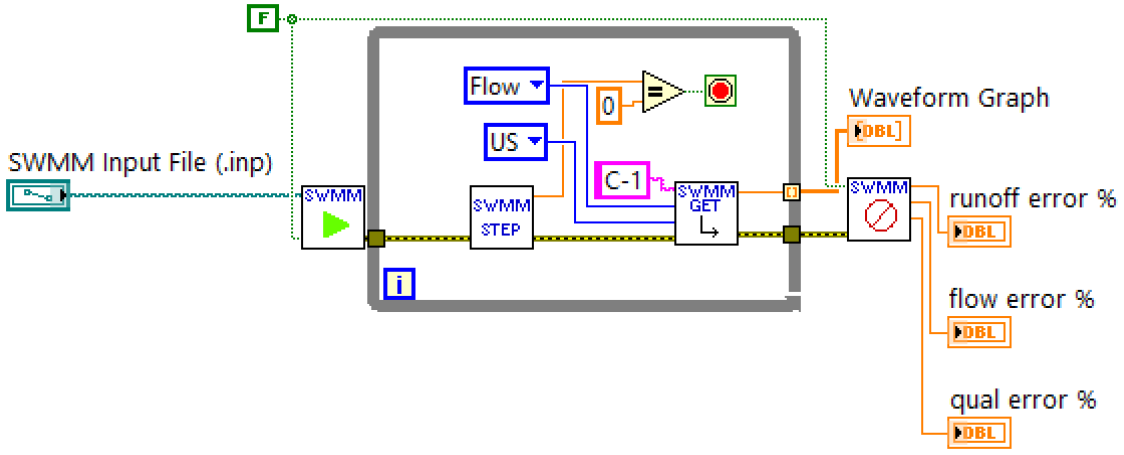


Figure 6. LabVIEW program example to reproduce a SWMM simulation, and extracting information.

An example of the structure of a LabVIEW program for running a simulation, modifying the setting of actuators, and extracting information from the SWMM model is shown below:

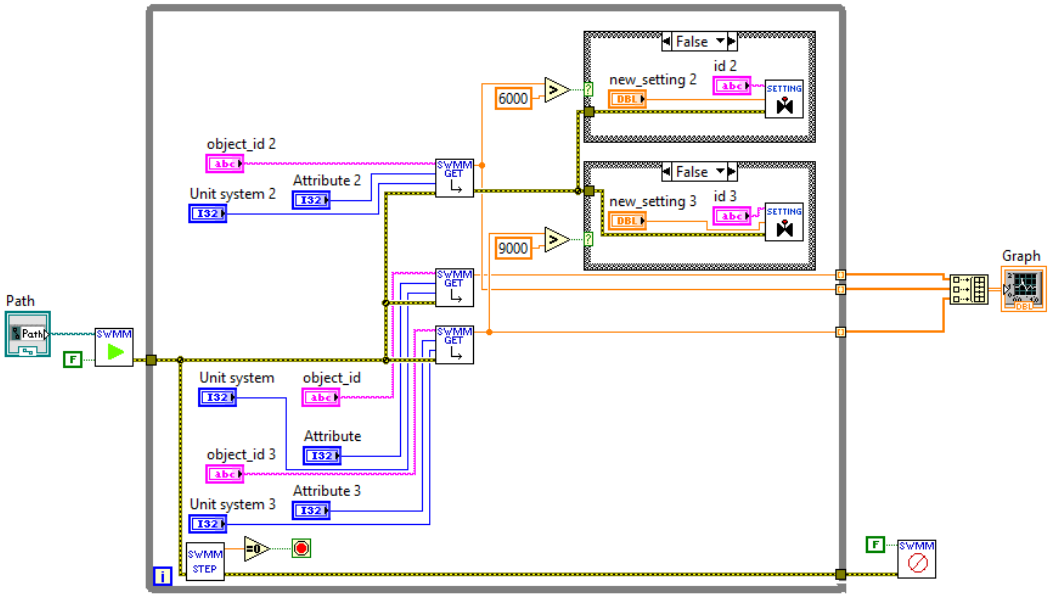


Figure 7. LabVIEW program example to reproduce a SWMM simulation, and extracting information.